DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9624

A Conceptual-Distance-Based Measurement Framework for Object-Oriented Specifications

by

Geert Poels

Guido Dedene

Katholieke Universiteit Leuven

Naamsestraat 69, B-3000 Leuven

ONDERZOEKSRAPPORT NR 9624

A Conceptual-Distance-Based Measurement Framework for Object-Oriented Specifications

by

Geert Poels

Guido Dedene

D/1996/2376/24

A CONCEPTUAL-DISTANCE-BASED MEASUREMENT FRAMEWORK FOR OBJECT-ORIENTED SPECIFICATIONS

Geert Poels
Guido Dedene

Katholieke Universiteit Leuven
Dept. of Applied Economic Sciences
Naamsestraat, 69
B-3000 Leuven
Belgium

E-mail Geert.Poels@econ.kuleuven.ac.be
Research Assistant of the Belgian National Fund for Scientific Research

Abstract - **To develop valid software measures we must know what attribute of the software entities is measured. However, in software engineering few attributes are really formally defined. This paper presents a measurement framework that defines and measures attributes as conceptual distances between software entities. Software entities are formally modelled as object types and conceptual schemes, and a number of measures of conceptual distance are proposed. These measures satisfy the metric properties of measure theory. It is further argued that this framework is the foundation of a new scientific software measurement approach.**

I. Introduction

According to measurement theorists a measure is a homomorphical mapping from an empirical relational system into a numerical relational system [9,21]. Hence, the measurement system contains:

- An **empirical relational system** (A,R) consisting of a set A of entities and a set R of relations on A as can be observed in reality. The relations in R order the entities of A according to the inherent quantity of a certain attribute of these entities.
- A **numerical relational system** (B,S) consisting of a set of numbers (e.g., the real numbers) and the usual ordering relations (e.g., $\leq$) on these numbers.
- A **measure** which is a mapping μ from (A,R) into (B,S) such that $\forall$ a, b $\in$ A: $\forall$ $R_i \in R$, $\exists$ $S_i \in S$: a R_i b $\Leftrightarrow$ μ(a) S_i μ(b). This condition is called the representation condition of ordinal measurement.

A software measure is a valid measure of an attribute of a software product, process or resource [11]. To validate a software measure (i.e., to show that it is a homomorphical mapping) we must know what the empirical relational system is. However, in software engineering it is mostly not known how this empirical relational system looks like [31]. Even a widely used measure like function points cannot be validated since it is not known what attribute of what entity is measured [1]. This is a pertinent problem in software measurement, especially the measures of internal attributes[1] of software entities.

The internal attribute that has received most attention is complexity. Literally dozens of measures were proposed that measure the 'complexity' of software. Fenton showed that there exists no such thing as the complexity of software [13]. Complexity is an attribute that can be defined according to many different viewpoints, e.g., structural complexity, psychological complexity, computational complexity, etc. But even for specific viewpoints numerous measures are proposed that purport to measure the same phenomenon, although the resulting measurement values are often non-monotonous (i.e., result in different rankings of the software products). As early as 1985 Zuse examined more than 50 measures of the structural complexity of software [30]. Most of them were not formally defined. Moreover, they measured a wide range of viewpoints for structural complexity.

To be able to define valid measures, we need to formally define the internal attributes of software entities. Until now, the approach that is mostly taken is to formulate a number of axioms for the attribute that must be measured [10,20,25,28]. Each measure of the attribute must satisfy these axioms. However it turned out that some of these research efforts resulted in inconsistent axiom sets, meaning that the axioms could not all be fulfilled simultaneously since they require different viewpoints to be measured [12,17]. Also, the axiomatic approach suffers from the flaw that the axioms proposed are necessary but not sufficient. Indeed, it is possible to develop measures that satisfy the axioms without being valid measures of the attribute [3].

Recently, Briand, Morasca and Basili extended the axiomatic approach to define properties for other internal attributes like size, length, coupling and cohesion [2]. Again, the property sets of what they call property-

[1] Attributes whose values depend only on the entity itself. The value of external attributes is determined by how the entity relates to its environment [11].

based software engineering measurement are not sufficient to guarantee valid measurement.

In this paper we go one step further by formally defining the attributes we wish to measure. The axiomatic approach showed that attribute definitions are very much needed, but also that it is very hard to find general accepted definitions. Consequently we believe it is not realistic to directly define these attributes. However, it should be possible to identify software entities having only a minimal amount (often nothing) of the attribute. These entities are called the null entities. Next, internal attributes are indirectly defined as conceptual differences between the entity to be measured and the null entity. The implicit hypothesis is that the more an entity differs from the null entity, the higher the value of the attribute.

Conceptual differences are equivalent to conceptual distances. If these conceptual distances can be measured, then the internal attributes are measurable, and valid software measures can be defined.

In this paper the conceptual-distance-based measurement framework is described and applied to the measurement of object-oriented specifications. In section 2 the framework is introduced. Next, in section 3 the development methodology is discussed that allows to formally model software entities and conceptual distances. Section 4 presents the measures of conceptual distance. Finally, section 5 contains the concluding remarks and focuses on further research directions.

II. The Formal Measurement Framework

Basically, the framework distinguishes three levels of measurement. At the first level conceptual distances between software entities are measured. The internal attributes and external attributes of software entities are measured at the second and third level respectively. Since software entities may differ along many dimensions, the conceptual distance is measured for each of these dimensions. If all dimensions are measured then we have in fact measured the global conceptual distance between the entities (figure 1).

LEVEL	ATTRIBUTE MEASURED
1	global conceptual distance (dimension 1, dimension 2, dimension 3, dimension 4, ...)
2	internal attributes
3	external attributes

Figure 1: Conceptual-Distance-Based Measurement Framework

Suppose two dimensions are distinguished. In figure 2 an hypothetical set of four software entities, including the null entity, is shown. Conceptual distances along the first dimension are graphically shown as solid lines. The conceptual distances along the second dimension are shown as broken lines. The global conceptual distance from one entity to another can be represented by a vector of dimension conceptual distances.

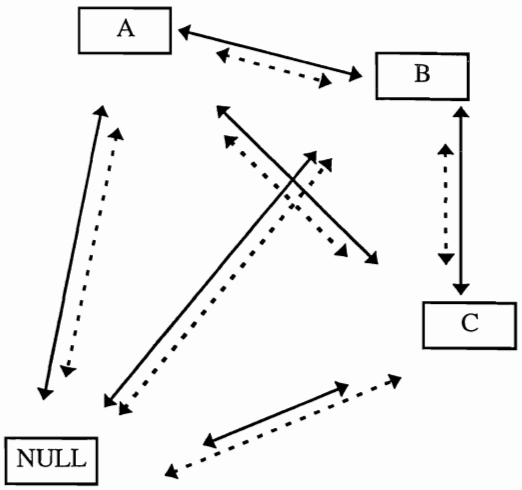


Figure 2: Conceptual distances between software entities

At the second level of measurement, internal attributes of the entities (e.g., size, functionality, reuse, ...) are indirectly defined and measured using the conceptual distances measured at the previous level. For each internal attribute the null entity, and the dimensions of conceptual distance that determine the value of the attribute must be identified. If more than one dimension is important global distances must be measured.

In figure 3 an internal attribute X is defined (and measured) as the conceptual distance along the first dimension from an entity to the null entity.

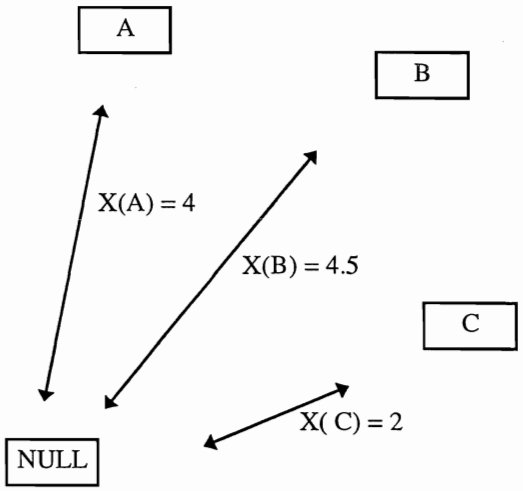


Figure 3: Measurement model internal attribute X

At the third and final level external attributes (e.g., maintainability, reusability, quality, ...) are measured. Since these attributes can only be quantified by considering the relationships between a software entity

and its environment, measurement models must be constructed that show how a measure of an external attribute is related to measures of internal attributes at the second level (or even to conceptual distances at the first level).

Figures 4-7 show measurement models comprising different types of measurable software-related entities. Entity type I (figure 4) is the kind of software entity described in the previous figures. Internal attributes are measured using the conceptual distance measures, and external attributes are measured using measures of conceptual distances, internal and external attributes of the same or other entities. Mostly, entities of type I are basic product entities like modules or object types. The measurements at the second and third level are mostly used to assess the values of the attributes of interest (e.g., what is the reuse level in a module?).

Type II entities (figure 5) are mostly software products that are themselves models or sets of other software products, for instance, programs that consist of modules, or conceptual schemes that consist of object types. For this kind of entities, conceptual distances can be indirectly measured in terms of the conceptual distances between their component elements. For instance, in section 4 we shall define the conceptual distance between object models as the average distance between the object types contained in these models. For type II entities measurement is mostly for assessment.

Type III entities (figure 6) are entities whose internal attributes are indirectly measured in terms of attributes of other entities. To this type belong software processes. Potentially useful internal attributes like development time or maintenance effort, or external attributes like costs can be predicted based on the measurement values of internal and external attributes of other software entities. For instance, the time needed to implement an OO design can be predicted based on the values of a number of attributes of the design (e.g., size, complexity, reuse, ...).

Type IV entities (figure 7) are entities whose external attributes are indirectly measured in terms of attributes of other entities. To this type belong software resources. For instance, the productivity of a programmer (an external attribute of a resource) can be assessed in terms of the size of the program he has written (an internal attribute of a product) and the time used for coding activities (an internal attribute of a process). Measurement for these types can also be used for prediction.

In the rest of this paper the framework is applied to object-oriented specifications. In the next section the development methodology used to formally describe software entities and attributes is discussed. Section 4 presents a set of measures of conceptual distances that can be positioned at the first measurement levels of business object types (entity type I) and business models (entity type II). We believe this set of measures constitutes a strong formal basis for the further measurement of software attributes. In subsequent research the other measurement levels will be filled with measures.

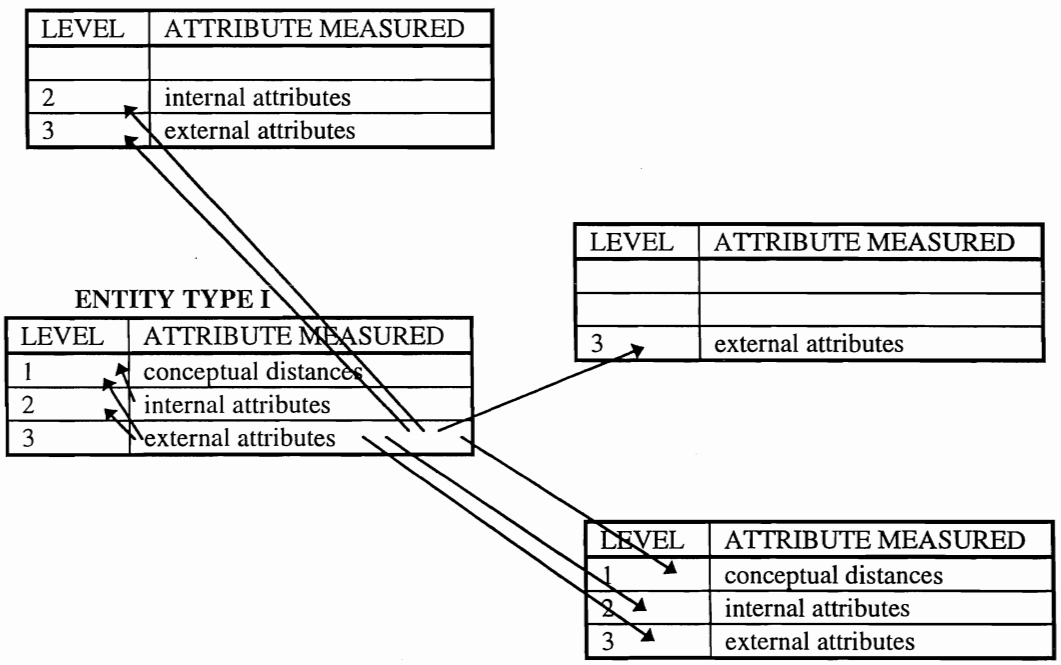


Figure 4: Conceptual-distance-based measurement model: software entity type I

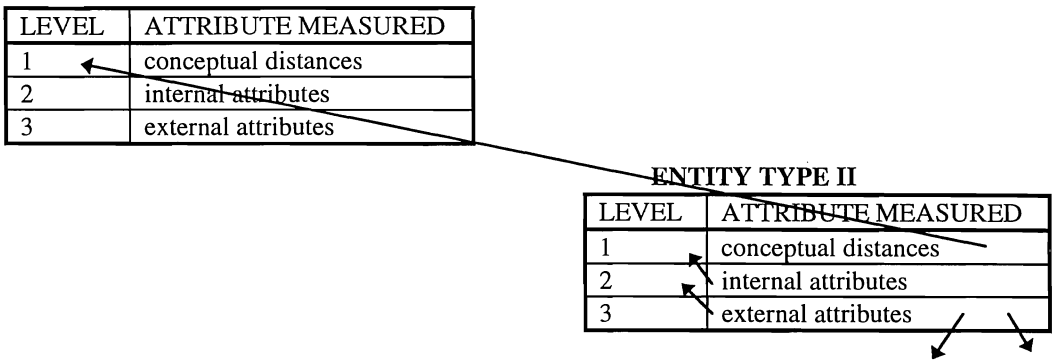


Figure 5: Conceptual-distance-based measurement model: software entity type II

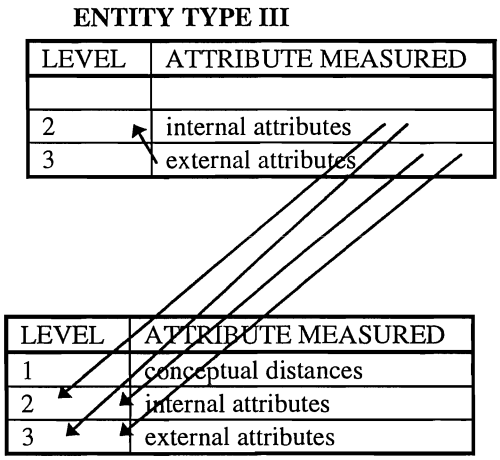


Figure 6: Conceptual-distance-based measurement model: software entity type III

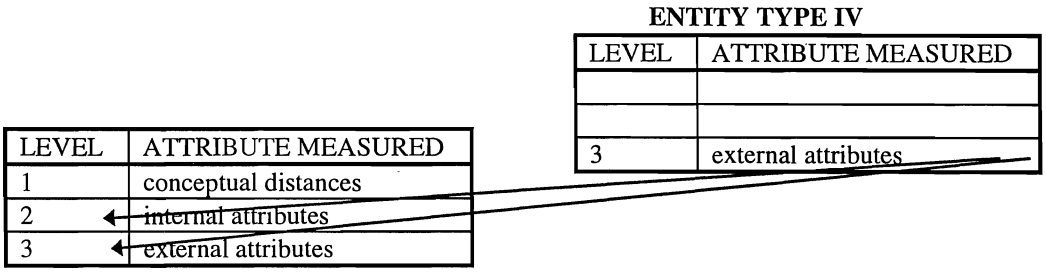


Figure 7: Conceptual-distance-based measurement model: software entity type IV

III. M.E.R.O.DE.

M.E.R.O.DE. is an acronym for Model-driven Entity-Relationship Object oriented DEvelopment [6,8,26,27]. It is a model-driven Object Oriented Analysis (OOA) method that models an information system in several layers, each representing a different level of abstraction. An overview of these different levels is given in figure 8, which is based on the Zachman framework for Application Development [29]. In this paper we are mainly concerned with the business and information model (also called design or function model) which are the two layers of the system specifications. The business model describes the exact functioning of the business in terms of object types, event types and the relationships between object and event types [6,8]. It models the most stable part of an information system, meaning the business itself. Around this core layer the information model is defined as the collection of functions that allows the interaction between business and users [8]. In the information model the information requirements of the users are described. This part of the system is much more

likely to change than the business core. That is the main reason for modelling these two different system abstractions separately.

Models required to represent the architecture of an information system	
Scope Model	model the scope and underlying strategy for an information system
Business Model	model the exact functioning of the business, show business entities, business constraints and business rules
Information Model	model the information functions for information input and output
Implementation Model	model the system as it is implemented in a particular technology

Figure 8: The Zachman Framework for Application Development [6,23,29]

In the framework of Cockburn M.E.R.O.DE. can be classified as a combinative approach to OOA [4]. In the business model the static parts are modelled by Object-Relationship Diagrams (figure 9), which are mainly based on Existence-Dependency Graphs and the traditional Entity-Relationship Diagrams [7]. The dynamic aspects are described by Jackson System Diagrams (figure 10) and Object-Event Tables (figure 11). The Object-Event Table identifies the relevant event types for each of the business object types and specifies which events create (C), destroy (D)or modify (M) object occurrences. For each object type a Jackson Structure Diagram describes the sequence restrictions imposed on the event types that are relevant for the object type. Apart from these techniques Abstract Data Types (figure 12) are used to further refine the business model by describing the attributes of the object types and how these attributes are modified if events happen [6]. Data constraints (e.g., referential integrity constraints) are modelled using a formal syntax (figure 13) [8].

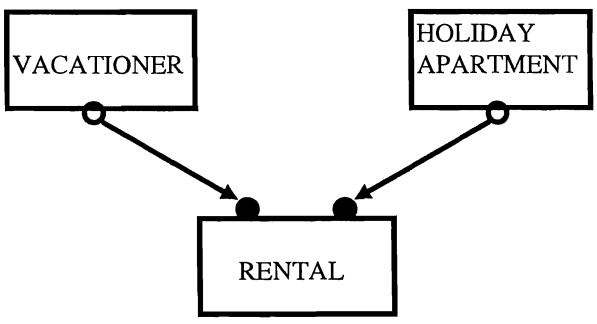


Figure 9: Object-Relationship Diagram

To guarantee the consistency between the different model views a process algebra was developed to formally model object types and conceptual schemes [8,23,24]. It is this additional aspect that differentiates M.E.R.O.DE. from most current methodologies for OOA. The formal modelling approach also allows us to define a formal measurement framework for object-oriented specifications [7]. In fact, without formal definitions of software entities and their attributes it is hard to show that software measures are valid [18].

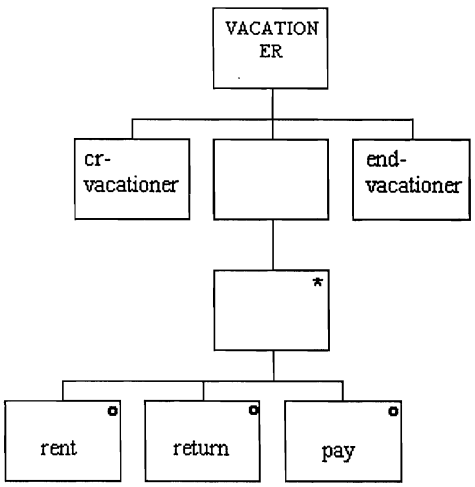


Figure 10 Jackson Structure Diagram for VACATIONER

The process algebra described in [24] was developed to formalise conceptual business models. The universe of event types in the model is denoted by the set A. The subset of A that is relevant for a certain object type (i.e., containing the event types in which the object type participates) is called the alphabet of the object type. The set P(A) contains all alphabets that can be constructed over A. The alphabet of an object type is selected by the function S_A.

Example

S_AVACATIONER = {cr-vacationer, end-vacationer, rent, return, pay}

S_AHOLIDAY APARTMENT = {cr-apartment, end-apartment, rent, return, pay}

S_ARENTAL = {rent, return, pay}

By means of the operators '.' (for sequence), '+' (for selection), and '*' (for iteration) regular expressions over A are built. Regular expressions describe the sequence constraints of object types by combining the events in the object type's alphabet through the operators. The Jackson Structure Diagrams are graphical representations of these regular expressions. The set R*(A) is the set of regular expressions over A. In [24] it is shown that R*(A),+,. is an idempotent semi-ring. The regular expression of an object type is selected by the function S_R.

Example

S_RVACATIONER = cr-vacationer . (rent + return + pay)* . end-vacationer

S_RHOLIDAY APARTMENT = cr-apartment . (rent + return + pay)* . end-apartment

S_RLOAN = rent . return . pay

To summarise, each business object type P is formally modelled by a tuple (α,e) over <P(A), R*(A)>, where $\alpha \subseteq$ A, e $\in$ R*(A), e $\neq$ 0 (i.e., different from deadlock) and all event types in the regular expression e must be contained in the alphabet α. Each business object type is further characterised by its attribute types and the data constraints on its attribute types. Let us introduce the selector functions S_S for the attribute set and S_D for the set of data constraints.

Example

S_SVACATIONER = {Vacationer-id, Vacationer-state}

S_SHOLIDAY APARTMENT = {Apartment-id, Apartment-state}

S_SRENTAL = {Rental-id, Rental-Vacationer-id, Rental-Apartment-id, Rental-state}

S_DVACATIONER = S_DHOLIDAY APARTMENT = {$\varnothing$}

S_DRENTAL contains all data constraints of figure 13

A business model is basically a set of object types M $\subseteq$ <P(A), R*(A)>, such that $\bigcup_{P \in M} S_A P = A$ [24].

Apart from this constraint a number of other restrictions guaranteeing the consistency of the dynamic and static aspects of the conceptual model, must be satisfied (see [24] for a detailed account). Also, research is being conducted to formally model the other abstraction levels in information system development, especially the information requirements level.

Object-Event Table	VACATIONER	HOLIDAY APARTMENT	RENTAL
CR-VACATIONER	C		
END-VACATIONER	D		
CR-APARTMENT		C	
END-APARTMENT		D	
RENT	M	M	C
RETURN	M	M	M
PAY	M	M	D

Figure 11: Object-Event Table

VACATIONER	HOLIDAY APARTMENT	RENTAL

State Vector
Vacationer-id, Vacationer-state
 	
Methods
CR-VACATIONER {
Vacationer-id :=
CR-VACATIONER_Vacationer-id;
Vacationer-state := "1";
}
END-VACATIONER {
Vacationer-state := "E";
}
RENT {
}
RETURN {
}
PAY {
}
	State Vector
	Apartment-id, Apartment-state
	
	Methods
	CR-APARTMENT {
	Apartment-id :=
	CR-APARTMENT_Apartment-id;
	Apartment-state := "1";
	}
	END-APARTMENT {
	Apartment-state := "E";
	}
	RENT {
	}
	RETURN {
	}
	PAY {
	}
		State Vector
		Rental-id, Rental-vacationer-id,
		Rental-apartment-id, Rental-state
		Methods
		RENT {
		Rental-id := RENT_rental-id;
		Rental-vacationer-id :=
		RENT_vacationer-id;
		Rental-apartment-id :=
		RENT_apartment-id;
		Rental-state := "1";
		}
		RETURN {
		Rental-state := "2";
		}
		PAY {
		Rental-state := "E";
		}

Figure 12: Abstract Data Types

```
END-VACATIONER:
        All RENTAL(Rental-vacationer-id =
                END-VACATIONER_Vacationer-id).Rental-state = "E"
END-APARTMENT:
        All RENTAL(Rental-apartment-id =
                END-APARTMENT_apartment-id).Rental-state = "E"
RENT:
        # RENTAL(Rental-apartment-id =
                RENT_apartment-id & Rental-state = "1") = 0
```

Figure 13: Data Constraints

IV. Measuring Conceptual Distances in M.E.R.O.DE. Specifications

Although in software measurement research many software metrics are proposed, few of them qualify as a metric. According to measure theory a metric is a function measuring the distance between two entities [5,15,22].

Definition

If X is a set of entities, then δ is a metric if and only if $\forall$ x, y, z $\in$ X:

$\delta(x,y) \geq 0$	(non-negativity)
$\delta(x,y) = 0 \Leftrightarrow x = y$	(identity)
$\delta(x,y) = \delta(y,x)$	(symmetry)
$\delta(x,y) \leq \delta(x,z) + \delta(z,y)$	(the triangle inequality)

In this section the function $\delta(P,Q)$ is proposed for measuring the conceptual difference between M.E.R.O.DE. business object types P and Q. Since $\delta(P,Q)$ satisfies the metric properties, it may be called a software metric without abusing existing mathematical concepts. Since a non-empty set M of M.E.R.O.DE. business object types and the distance function $\delta(P,Q)$: MxM $\rightarrow$ Re is said to form a metric space [15], it is common to call the conceptual difference between P and Q the conceptual distance from P to Q.

In section 3 it was described that M.E.R.O.DE. business object types are composed of an alphabet of event types, a regular expression on these event types describing the sequence constraints, a set of attribute types, and eventually, some data constraints. To compare object types each of these aspects must be considered, i.e., object types can differ along each of these four dimensions. Since it does not seem possible at this moment to measure differences along the four dimensions simultaneously, first a number of pseudo-metrics are developed to measure conceptual distances on each of these dimensions separately. A pseudo-metric satisfies a weaker kind of identity axiom ($x = y \Rightarrow \delta(x,y) = 0$, but not $\delta(x,y) = 0 \Rightarrow x = y$), meaning that if two entities do not differ on one dimension, they are not necessarily the same for all dimensions [15].

Next, the pseudo-metrics are combined to define the metric $\delta(P,Q)$ measuring the global conceptual distance between object types. Also, $\delta(P,Q)$ is used to define another metric $\delta_M(P,Q)$ measuring the conceptual distance between conceptual schemes.

A. Pseudo-metrics

In order to derive valid measures for the conceptual distance dimensions we take the Model-Order-Mapping (MOM) approach of Gustafson, Tan and Weaver [14]. This approach is depicted in figure 14.

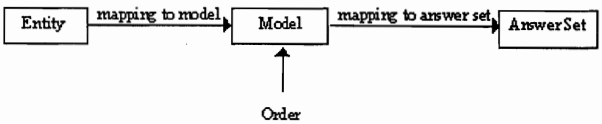
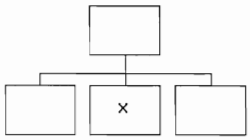
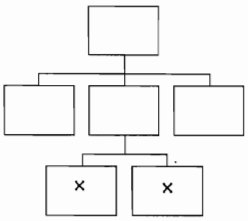
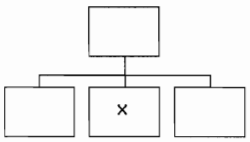
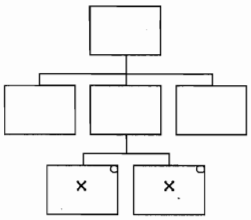
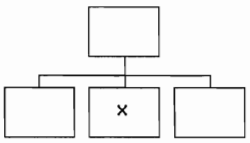
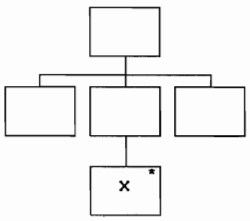
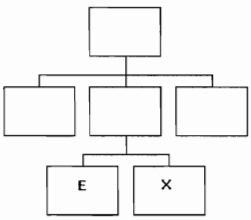
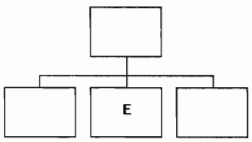
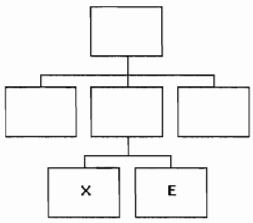
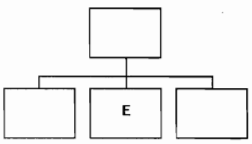
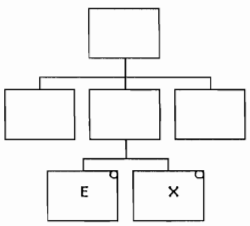
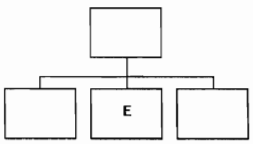
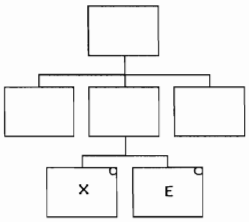
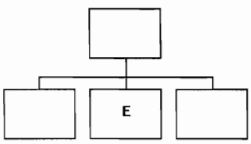


Figure 14: Model-Order-Mapping approach

- **Entity**
  The entity that is measured is a pair of business object types (P,Q) having a number of attributes, such as the dimensions of conceptual distance. For each of the attributes we wish to measure a model must be specified that captures the attribute of interest while hiding the other attributes. This model should be mathematically defined.
- **Mapping from Entity to Model**
  The mapping from the entity to the model must be formally defined. In M.E.R.O.DE. these mappings are the selector functions S_A, S_S, S_D and S_R.
- **Model**
  A formally derived model allows to identify orders on the set of entities. For the alphabet, attribute set and data constraint dimensions each model consists of a pair of sets (S_iP, S_iQ), where S_i is replaced by S_A, S_S or S_D respectively. Measuring the conceptual distance in alphabet, attribute set and data constraints from P to Q amounts for each pair of sets to the measurement of the difference between the sets.
  The selector function S_R does not result in a set. For each object type P, the function S_RP selects the regular expression that is the mathematical formalism of the sequence constraints that apply to P. The operands of these regular expressions are the business event types in which the object type participates. The operators are the sequence (.), selection (+) and iteration (*) operators. Since differences between the

transformation	from	to
t_1: x → x.x		
t_2: x → x+x		
t_3: x → x*		
t_4: e.x → e		
t_5: x.e → e		
t_6: e+x → e		
t_7: x+e → e		

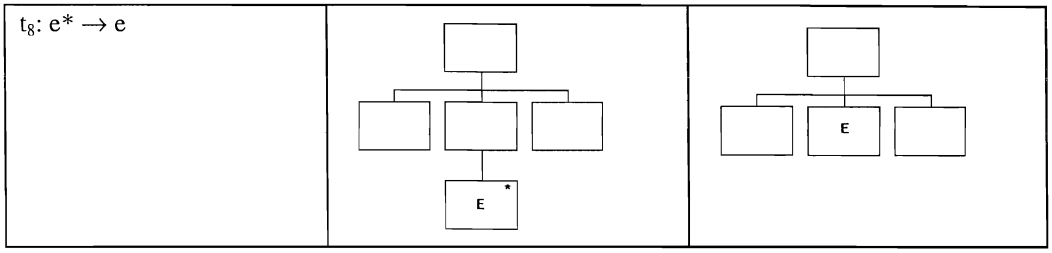


Figure 15: Allowable transformations on $\setminus_{seq}$-projected sequence constraints

In [19] it is shown that these pseudo-metrics are measured on a ratio scale[3].

Figure 17 summarises the development of the pseudo-metrics. Note that in the original Model-Order-Mapping approach the order that was imposed on the models was only a partial order[4] [14]. Since a partial order is not strongly complete the mappings are not homomorphic meaning that the representation condition is not satisfied in the $\Leftarrow$ direction.

Apart from a measure's definition, procedures are needed that lead to consistent measurement of the attributes of an entity. These procedures are part of what Kitchenham et al. call the measurement protocol [16]. This protocol contains the model of the entity on which measurement is based, and the procedures, guidelines and rules to carry out the actual measurement.

As far as models are concerned, they are already identified using the MOM approach. However, the specific measurement procedures for each pseudo-metric were not described. They are nonetheless important because actual measurement should be independent of environment and the person carrying out the measurement.

[3] The fact that we have a weak order only guarantees an ordinal scale. However, the metric axioms impose a ratio scale on the measures since the class of admissible transformations of scale (i.e., those transformations that do not alter the properties of the scale) is limited to scalar multiplications (see figure 16 for an overview of scale types).

class of admissible transformations	scale type
$\phi(x)=x$ (identity)	absolute
$\phi(x)=a.x$, $\forall a > 0$ similarity transformation	ratio
$\phi(x)=a.x+b$, $\forall a > 0$ positive linear transformation	interval
$x \leq y \Leftrightarrow \phi(x) \leq \phi(y)$ monotone increasing transformation	ordinal
any one-to-one ϕ	nominal

Figure 16: Scale types[21]

[4] Partial orders are reflexive ($\forall$ a : a $\leq_i$ a), antisymmetric ($\forall$ a, b : a $\leq_i$ b and b $\leq_i$ a $\Rightarrow$ a = b) and transitive ($\forall$ a, b, c : a $\leq_i$ b and b $\leq_i$ c $\Rightarrow$ a $\leq_i$ c).

In this paper we are more concerned about the conceptual definition of the measures than we are interested in their measurement protocols. Anyway, most pseudo-metrics can be straightforwardly calculated.

Example

1. The conceptual distances in alphabet are:
δ_{alph}(VACATIONER, HOLIDAY APARTMENT) = 4
δ_{alph}(VACATIONER, RENTAL) = 2
δ_{alph}(HOLIDAY APARTMENT, RENTAL) = 2

2. The conceptual distances in attribute set are:
δ_{atr}(VACATIONER, HOLIDAY APARTMENT) = 4
δ_{atr}(VACATIONER, RENTAL) = 4
δ_{atr}(HOLIDAY APARTMENT, RENTAL) = 4
(Note that we considered Vacationer-id = Rental-vacationer-id and Apartment-id = Rental-apartment-id)

3. The conceptual distances in $\setminus_{seq}$-projected sequence constraints are:
δ_{seq}(VACATIONER, HOLIDAY APARTMENT) = 0
δ_{seq}(VACATIONER, RENTAL) = 3
δ_{seq}(HOLIDAY APARTMENT, RENTAL) = 3

4. The conceptual distances in data constraints are:
δ_{data}(VACATIONER, HOLIDAY APARTMENT) = 0
δ_{data}(MEMBER, RENTAL) = 3
δ_{data}(HOLIDAY APARTMENT, RENTAL) = 3

B. Metrics

The pseudo-metrics presented in the previous subsection can be used to define a number of indirect measures of conceptual distance. First of all, a measure of the global conceptual distance between business object types is defined as a vector of the pseudo-metrics.

Definition

$\forall$ P, Q $\in$ M: $\delta(P,Q) = (\delta_{alph}(P,Q), \delta_{atr}(P,Q), \delta_{seq}(P,Q), \delta_{data}(P,Q))$

Each vector component measures the conceptual distance between object types P and Q along a single dimension. According to the formal M.E.R.O.DE. model of a business object type, if the distance along each dimension is measured, we have in fact measured the global conceptual distance. The vector $\delta(P,Q)$ may be called a metric, since it satisfies a number of axioms very similar

conceptual distance dimension (attribute measured)	alphabet	attribute set	data constraints	sequence constraints
software entity measured	pair of business object types (P,Q)	pair of business object types (P,Q)	pair of business object types (P,Q)	pair of business object types (P,Q)
mapping functions to measurement model	S_A	S_S	S_D	S_R, $\backslash_{seq}$
measurement model	S_AP, S_AQ	S_SP, S_SQ	S_DP, S_DQ	$S_RP\backslash_{seq}$, $S_RQ\backslash_{seq}$
ordering relation	$\leq_{alph}$	$\leq_{atr}$	$\leq_{data}$	$\leq_{seq}$
mapping to answer set	δ_{alph}	δ_{atr}	δ_{data}	δ_{seq}
pseudo-metric definition	$\delta_{alph}(P,Q) = \lvert S_AP \Delta S_AQ \rvert$	$\delta_{atr}(P,Q) = \lvert S_SP \Delta S_SQ \rvert$	$\delta_{data}(P,Q) = \lvert S_DP \Delta S_DQ \rvert$	$\delta_{seq}(P,Q)$ = minimum number of transformations to transform $S_RP\backslash_{seq}$ into $S_RQ\backslash_{seq}$
answer set	(Re, ≤)	(Re, ≤)	(Re, ≤)	(Re, ≤)

Figure 17: Measuring conceptual distances with pseudo-metrics (level 1 of the conceptual measurement framework)

to the metric axioms [19]. Note that $\delta(P,Q)$ satisfies a stronger version of the identity axiom than its composing pseudo-metrics, because if $\delta(P,Q) = (0,0,0,0)$ we may conclude that P = Q, since all aspects of difference are accounted for.

The fact that δ is similar to a metric does not imply that the scale type of the function is ratio. In [30] a number of indirect measures of control flow complexity that are based on pairs of direct complexity measures are validated. Zuse asserts that if $\leq_c$ is an empirical relation meaning 'is less or equally complex than', and P and P' are programs, then

$$P \leq_c P' \Leftrightarrow (\mu_1(P), \mu_2(P)) \leq (\mu_1(P'), \mu_2(P'))$$
$$\Leftrightarrow \mu_1(P) \leq \mu_1(P') \wedge \mu_2(P) \leq \mu_2(P')$$

If we extend this reasoning to vectors, then a vector A is smaller than or equal to a vector B whenever every vector component value a_i in A is smaller than or equal to every corresponding vector component value b_i in B. However for arbitrary vectors A and B, there is a non negligible chance that neither $A \leq B$, nor $B \leq A$.

For the object types P, Q, R, S $\in$ M, the conceptual distance from P to Q is less than or the same as the conceptual distance from R to S (hereafter written as $(P,Q) \leq_{cd} (R,S)$) if and only if $\delta(P,Q) \leq \delta(R,S)$. If δ is a vector, then $\delta(P,Q) \leq \delta(R,S) \Leftrightarrow \delta_i(P,Q) \leq \delta_i(R,S)$ for i = alph, atr, seq and data. Again, it is clear that uncomparabilities can arise. The binary relation $\leq_{cd}$ is reflexive and transitive. Hence, we have a quasi order [21]. It is however not strongly complete since it is not always true that $\delta(P,Q) \leq \delta(R,S)$ or $\delta(R,S) \leq \delta(P,Q)$. Thus, there is no weak ordering. Even the property of antisymmetry necessary for a partial ordering, is not satisfied.

According to Zuse, indirect measures based on a vector have a scale type called half-ordering scale [30]. A half-ordering scale is somewhere between the nominal and the ordinal scale types in the scale hierarchy. This simply means that although the global conceptual distance of some pairs of object types can be compared, not all of them are comparable. So, using a half-ordering scale, pairs of object types can be ranked, but many rankings are possible, and none of them will comprise all object type pairs.

Example

δ(VACATIONER, HOLIDAY APARTMENT) = (4, 4, 0, 0)
δ(VACATIONER, RENTAL) = (2, 4, 3, 3)
δ(HOLIDAY APARTMENT, RENTAL) = (2, 4, 3, 3)
According to these measurement values we may only conclude that (VACATIONER, RENTAL) $\leq_{CD}$ (HOLIDAY APARTMENT, RENTAL) and that (HOLIDAY APARTMENT, RENTAL) $\leq_{CD}$ (VACATIONER, RENTAL). However, since each vector component is measured on a ratio scale, all pairs of object types may be compared using the empirical relations $\leq_{alph}$, $\leq_{atr}$, $\leq_{data}$ and $\leq_{seq}$.

A second indirect measure of conceptual distance based on the pseudo-metrics is the function δ_M measuring the conceptual distance between business models. Recall that in M.E.R.O.DE. business models are essentially sets of object types $M \subseteq \langle P(A), R^*(A) \rangle$ satisfying a number of restrictions [24]. If M_P and M_Q are business models, then we define the function $\delta_M(M_P,M_Q)$ as:

$$= (0,0,0,0) \qquad \Leftrightarrow M_P \Delta M_Q = \varnothing$$

$$= \left(\frac{\sum_{i=1}^{I}\sum_{j=1}^{J} \delta_{alph}(P_i,Q_j)}{I.J}, \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} \delta_{atr}(P_i,Q_j)}{I.J}, \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} \delta_{seq}(P_i,Q_j)}{I.J}, \frac{\sum_{i=1}^{I}\sum_{j=1}^{J} \delta_{data}(P_i,Q_j)}{I.J} \right) \Leftrightarrow M_P \Delta M_Q \neq \varnothing$$

where:
M_P and M_Q are non-empty dynamic conceptual schemes (i.e., formally defined business models);
$M_P \Delta M_Q = \varnothing \Leftrightarrow$
$(\forall P \in M_P, \exists Q \in M_Q : \delta(P,Q) = (0,0,0,0)) \wedge$
$(\forall Q \in M_Q, \exists P \in M_P : \delta(Q,P) = (0,0,0,0))$;
cardinality $(M_P) = I$;
cardinality $(M_Q) = J$;
$P_i \in M_P \quad i = 1, ..., I$;
$P_j \in M_Q \quad j = 1, ..., J$;

In [19] it is shown that δ_M satisfies properties similar to the metric axioms. The scale type of the measure is half-ordering.

The measure of distance δ_M is equal to the average distance between the object types of two different, non-empty dynamic conceptual schemes. A large average distance between object types leads to a large difference between the dynamic conceptual schemes. If the schemes are not different, then the measure is zero by definition. The identity axioms are by definition true. The schemes may however not be empty. For empty schemes the measure cannot be calculated. We believe this restriction will not hamper the usability of the measure.

V. Conclusions and further research

In this paper new strategies were explored for software measurement. A conceptual framework was proposed that distinguishes three levels of measurement. At the first level conceptual differences between software entities are assessed by mathematical functions satisfying the metric and pseudo-metric properties of measure theory. At the next level potentially useful internal attributes (e.g., size, complexity, functionality) of the entities are defined and indirectly measured as conceptual differences between these entities and a null entity. Finally, at the third level the measures of the underlying levels are used to indirectly assess and predict external product, resource and process attributes such as reusability, productivity and development effort.

This paper described the general measurement framework and applied it to object-oriented specifications developed with M.E.R.O.DE., an object-oriented development methodology that allows to formally model object types and conceptual schemes. A number of software pseudo-metrics were proposed that are used at the direct measurement level to measure conceptual distances between business object types. Also the measurement of differences between conceptual schemes was considered.

Further research will extend the existing set of measures for M.E.R.O.DE. specifications in three directions:

- In-depth by formulating measurement protocols for each of the pseudo-metrics.
- Horizontally by developing measures of conceptual distance for other kinds of object types and models (e.g., for information and function object types, and for the information model in M.E.R.O.DE.).
- Vertically by formally defining and measuring other relevant attributes of M.E.R.O.DE. products, processes and resources in terms of conceptual distances.

Although the measures proposed in this paper are specifically developed for M.E.R.O.DE., we believe the measurement framework is generic and can be applied to any development methodology that models relationships, abstract data types, constraints, Finite State Machines (mathematically equivalent with Jackson Structure Diagrams and regular expressions [20]), etc. Of course, the more formal the methodology, the easier the framework is applied.

We hope that this new approach can help to solve some of the current problems in software measurement, especially those problems related to attribute and measure definitions and measure validation.

VI. References

[1] Abran A. and P.N. Robillard, 'Function Points: A Study of Their Measurement Processes and Scale Transformations', *Journal of Systems and Software*, Vol. 25, 1994, pp. 171-184.

[2] Briand L., S. Morasca and V. Basili, 'Property-Based Software Engineering Measurement', *IEEE Transactions on Software Engineering*, Vol. 22, No. 1, 1996, pp. 68-68.

[3] Cherniavski J.C. and C.H. Smith, 'On Weyuker's Axioms For Software Complexity Measures', *IEEE Transactions on Software Engineering*, Vol. 17, No. 6, 1991, pp. 636-638.

[4] Cockburn A.A.R., 'The impact of object-orientation on application development', *IBM Systems Journal*, Vol. 32, No. 5, 1993, pp. 420-444.

[5] DeBarra G., *Introduction to Measure Theory*, Van Nostrand Reinhold Company, London, 1974, 287 pp.

[6] Dedene G. and M. Snoeck, 'M.E.R.O.DE.: A Model-driven Entity-Relationship Object-oriented Development method', *ACM SIGSOFT Software Engineering Notes*, Vol. 19, No. 3, 1994, pp. 51-61.

[7] Dedene G. and M. Snoeck, 'Flexible function modelling around object-oriented business models', presented at Object Technology 95, Oxford, UK, March 1995.

[8] Dedene G. and M. Snoeck, 'Formal Deadlock Elimination in an Object-Oriented Conceptual Schema', *Data & Knowledge Engineering*, Vol. 15, 1995, pp. 1-30.

[9] Ellis B., *Basic Concepts of Measurement*, Cambridge University Press, 1968, 220 pp.

[10] Fenton N.E. and R.W. Whitty, 'Axiomatic approach to Software Metrication through Program Decomposition', *The Computer Journal*, Vol. 29, No. 4, 1986, pp. 330-339.

[11] Fenton N.E., *Software Metrics, A Rigorous Approach*, Chapman & Hall, London, 1991, 337 pp.

[12] Fenton N.E., 'When a Software measure is not a measure', *Software Engineering Journal*, Vol. 7, No. 5, 1992, pp. 357-362.

[13] Fenton N.E., 'Software Measurement: A Necessary Scientific Basis', *IEEE Transactions on Software Engineering*, Vol. 20, No. 3, 1994, pp. 199-206.

[14] Gustafson D.A., J.T. Tan and P. Weaver, 'Software Measure Specification', *ACM Software Engineering Notes*, Vol. 18, No. 5, 1993, pp. 163-168.

[15] Kingman J.F.C. and S.J. Taylor, *Introduction to Measure and Probability*, Cambridge University Press, 1966, 401 pp.

[16] Kitchenham B., S.L. Pfleeger and N.E. Fenton, 'Towards a Framework for Software Measurement Validation', *IEEE Transactions on Software Engineering*, Vol. 21, No. 12, 1995, pp. 929-944.

[17] Melton A.C., D.A. Gustafson, J.M. Bieman and A.L. Baker, 'A mathematical perspective for software measures research', *Software Engineering Journal*, Vol. 5, No. 5, 1990, pp. 246-254.

[18] Poels G. and G. Dedene, 'Formal Measurement in Object-Oriented Software', presented at Object Technology 96, Oxford, UK, March 1996.

[19] Poels G. and G. Dedene, 'Formal Software Measurement for Object-Oriented Business Models', *Proceedings 7th European Software Control and Metrics Conference*, Wilmslow, UK, May 1996.

[20] Prather R.E., 'An Axiomatic Theory of Software Complexity Measure', *The Computer Journal*, Vol. 27, No. 4, 1984, pp. 340-347.

[21] Roberts F. S., *Measurement Theory with Applications to Decisionmaking, Utility, and the Social Sciences*, Addison-Wesley Publishing Company, Reading, 1979, 420 pp.

[22] Royden H.L., *Real Analysis*, Macmillan Publishing Company, New York, 1968, 349 pp.

[23] Snoeck M., 'Formele specificaties: de basis voor kwaliteit', *Informatie*, Vol. 36, No. 4, 1994, pp. 257-266, (*in Dutch*).

[24] Snoeck, M., *On a Process Algebra Approach for the Construction and Analysis of M.E.R.O.DE.-Based Conceptual Models*, Phd dissertation, departement Computerwetenschappen, K.U.Leuven, 1995, 209 pp.

[25] Tian J. and M.V. Zelkowitz, 'A Formal Program Complexity Model and Its Application', *Journal of Systems and Software*, Vol. 17, 1992, pp. 253-266.

[26] Verhelst M., *Objectgerichte Systeemontwikkeling: een praktische aanpak met JSD en M.E.R.O.DE.*, Kluwer, Deventer, 1992, (*in Dutch*).

[27] Verhelst M., 'Model Based Entity-Relationship Object Oriented Development (M.E.R.O.DE.)', *Beleidsinformatica Tijdschrift*, Vol. 20, No. 4, 1994, 51 pp.

[28] Weyuker E.J., 'Evaluating Software Complexity Measures', *IEEE Transactions on Software Engineering*, Vol. 14, No. 9, 1988, pp. 1357-1365.

[29] Zachman J.A., 'A framework for information architecture', *IBM Systems Journal*, Vol. 26, No. 3, 1987, pp. 276-292.

[30] Zuse H., *Messtheoretische Analyse von statischen Softwarekomplexitätsmassen*, Phd dissertation, Fachbereich Informatik, Technische Universität Berlin, 1985, 414 pp.

[31] Zuse H. and P. Bollmann, 'Software Metrics, Using Measurement Theory to Describe the Properties and Scales of Static Software Complexity Metrics', *ACM Sigplan Notices*, Vol. 24, No. 8, 1989, pp. 23-33.